AUSTRALIAN OS9 NEWSLETTER

EDITOR	Gordon Bentzen	(07) 344-3881
SUB-EDITOR	Bob Devries	(07) 372-7816
TREASURER	Don Berrie	(07) 375-1284
LIBRARIAN	Jean-Pierre Jacquet	(07) 372-4675
SUPPORT	Brisbane OS9 Users Group	

Addresses for Correspondence

Editorial Material:

Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

Subscriptions & Library Requests:

Jean-Pierre Jacquet
27 Hampton Street
DURACK Qld 4077

Volume 5 April 1991 Number 3

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group
Volume 5 Number 3

EDITOR : Gordon Bentzen
SUBEDITOR : Bob Devries
TREASURER : Don Berrie
LIBRARIAN : Jean-Pierre Jacquet
SUPPORT : Brisbane OS9 Level 2 Users Group.

Here is yet another appeal for contributions to our newsletter. Can you help to fill these pages with something which may be of interest to other National OS9 Usergroup members?? We would be happy to include your questions here so that someone may be able to help with a solution.

Our membership continues to grow with some 53 members spread across every State in Australia as well as Papua New Guinea, Germany and the United States. The predicted move to OSK has not as yet been widespread within Australia with the majority of our members still happy to run OS9 on the Tandy CoCo family, but who knows what the future will bring. I know of one of our usergroup members who has placed an order for the MMI but I won't mention his name at this point.

OS9 continues to hold our interest despite the fact that it is far from the mainsteam operating system. We all have had, no doubt, some exposure to MS-DOS, and some would say far too much exposure for their liking, but we still come back to OS9 because we miss the many features of this powerful operating system which are non-existent in other systems. OS9 and OSK can be "fun" as well as frustrating at times, particularly when things "go wrong", "go wrong". However, help is always available from other OS9'ers.

One of our members from Canberra has recently added a hard drive to his CoCo utilising the Burke & Burke interface. Now when one undertakes a project like this, one can expect to run into a few problems along the way, mainly because we are suddenly faced with unfamiliar software and procedures. Well our man in Canberra did run into a couple of problems which prevented him from being able to format the newly added hard drive, and he had reached the point of complete frustration when he decided to contact me. Don Berrie was also dragged into this problem (not that Don needs much dragging whenever an OS9 problem appears) and we were able to help out. The hard drive is now formatted under OS9, and we were advised by our Canberra member that he had tracked down the source of his problem. So here is a tip; when building a device descriptor for a hard drive, DO NOT set the "park track" as track zero, even if the hard drive in question does not need to be "parked" or has "auto-park".

This is just one example of how we can help each other and this experience may also help someone else, we hope so.

So what is your particular problem with OS9? Or perhaps you can pass on a few valuable tips from your own experiences.

PROBLEM: I have used the screendump utility "SPrint" in assembler from the P.D. library which is from Ron Lammardo. This utility dumps a text screen from a specified window to "std out" by default, or allows redirection to the printer. Now when I redirect output to my printer, the "_" underscore character $5F is ignored and the "^" $5E is printed as "↑" in reverse. Help!!

IN THIS EDITION:-

Bob Devries has included C source for the MAN utility a la UNIX MAN which allows .doc or manual files to be listed. This is a very useful utility for hard drive users, and could also be handy for a floppy based system if all "manual" files were stored on one disk.

Don Berrie has included some more info files from the CoCo list server which helps us to keep up to date with developments overseas.

We continue to work on the long promised database of the public domain library, so don't give up hope.

Cheers, Gordon.

CoCo-Link

CoCo-Link is an excellent magazine to help you with the RSDOS side of the Colour Computer. It is a bi-monthly magazine published by Mr. Robbie Dalzell. Send your subscriptions to:

CoCo-Link
31 Nedlands Crescent
Pt. Noarlunga Sth.
South Australia
Phone: (08) 3861647

Getting SAS-sy With Your Disks
by Pete Lyall

Whether folks are finding hard disks to be more affordable, or mandatory for increased storage capabilities, some combination of the above is causing more of them to appear on Coco3 (and other) OS9 systems. With all this space, power, and capability comes the the inevitable list of questions and confusing options. I won't attempt to deal with them all here, but I will try to clear the air on a couple of device and path descriptor options that seem to be giving some new hard disk owners a bit of a tough time - DD.BIT (number of sectors per cluster) and IT.SAS (minimum sector allocation size).

Since the DD.BIT is a bit easier to cover, let's do that first to get warmed up for the other stuff. Another good name for DD.BIT might well be 'clustersize'. In order to understand what clustersize is, we're going to have to dig a little bit into the layout of the disk. All disks, be they floppy or hard disks, have at least three things on them when they are initially formatted. These are:

o - Logical Sector Number Zero (LSN0)

o - Disk Bitmap

o - Root Directory

The LSN0 is essentially a sector containing all the information about the disk's personality. It contains information on how many tracks and sectors the disk has, and also how many sides (or heads, in the case of a hard disk) are available. Other information includes the time that this volume (disk) was created or formatted, who the owner is, where and how large the bootfile is (if any), the disk name, and lastly the bitmap information. This bitmap information tells the system how many bytes are in the bitmap, and what the size is in sectors (clustersize) of the space that each bit in the map represents. More on this in a minute.

The bitmap is basically a map of the space available on a disk. Each bit of each byte in the bitmap represents a block of space on the disk. One analogy to the bitmap would be the seating chart you might see in a restaurant when you are in the reception area. Tables that are marked on the chart with an 'X', or possibly illuminated by an LED, indicate that the table is in use.

Tables that are not explicitly marked are available to be used. Much is the case with the bitmap. If a bit is on (in the '1' state), it tells the system that the block of disk space that it represents is in use. What the space is used for is of no consequence to the bitmap. It could be part of a file, a directory, or even part of the bitmap itself. Conversely, an off bit ('0') indicates that the space is available for whatever needs it.

Notice that I have managed to avoid defining how big this space is. The reason? It is variable. Each bit in the bitmap represents one cluster. The size of this cluster is defined in DD.BIT in LSN0, which is read by the system before any other manipulation of the disk is done. A cluster may be from 1 to $FFFF (65535) sectors in length.

Ready for a surprise? Almost everyone uses a cluster size of 1. So why bother with all this confusing talk of clusters and sectors? There was a time in OS9 history when lower density floppies were the predominant storage device. I'm going to hazard a guess and say that early os9 programmers decided to limit the length of the bitmap to 1 sector, thinking they wouldn't need any more. After all, if we consider the case of clustersize = 1 sector, and a formula of:

(TRACKS*SECTORS/BITSperBYTE)/CLUSTERSIZE = BITMAP SIZE

a single-sided 35 track, double density drive at 16 sectors per track (standard os9 format) only required 70 bytes of bitmap space (35*16/8)/1, and a single-sided 77 track 8 inch disk, at 26 sectors per track would fit into 251 bytes of bitmap space (77*26/8)/1.

The problems came with drives of higher capacity. A conventional double sided 80 track drive would overflow the bitmap sector ((80*16*2/8)/1 = 320 bytes). You can easily imagine how a hard disk might affect this situation. Enter clustersize. If the clustersize is set to 2, each bit in the bitmap now represents 2 sectors of disk space. In the case of the 80 track floppy above, this alters the formula to ((80*16*2/8)/2 = 160 bytes). At a clustersize of 2 sectors, the bitmap can represent roughly 1 megabyte of space.

Obviously this is not large enough for even the smallest hard disks. The cluster size would have to be raised to 10 sectors to represent a 5 megabyte drive in a one sector bitmap. For a 20 megabyte drive, each bit in the bitmap would have to represent 40 sectors, and it gets progressively worse with larger drives. This is where we can start to see the flaw in this design. Suppose you need to create a very short procedure file that were to contain the command:

```
echo This is a test >/t2
```

This file should require only about 25 bytes including the carriage return. Because the hardware prevents us from using anything smaller than a sector, our file size will be 25 bytes, but the disk space used should be 1 sector. Assuming that we had done this on the 20 megabyte drive described above, we would actually have used 40 sectors of disk space (10 kbytes) - a waste of 39 sectors.

Why? Because the minimum chunk that we could mark as used or available in the bitmap was 40 sectors. When we marked the area as in-use, we also marked the rest of the cluster as in use. What a waste!

Fortunately, Microware felt the same way and altered the bitmap design. Now the bitmap may be more than one sector in length. In fact, it may be up to 65535 bytes long (8192 sectors). The number of bytes in the bitmap is also specified in LSN0 in DD.MAP. If clustersize were 1 sector, this bitmap would be large enough to service a 130 megabyte hard disk.

For larger disks, the clustersize has to be raised to accommodate the capacity. A clustersize of 8, for instance, could handle more than 1 gigabyte of storage. The current bitmap size and clustersize ceiling is capable of supporting a disk that contains 1,073,741,824 sectors, or approximately 256 gigabytes. I think it should last us for a while, eh?

That's probably all you'll ever wanted to know (or more) about clustersize. For almost all practical purposes, it should be left at a size of 1 sector per cluster. This is sufficient to support those of us with disks that are 130 megabytes or smaller.

The 'IT.SAS', or segment allocation size, is used to determine how much of the disk's real estate is given up each time a request for more disk space is made. This item is located in the device descriptor for each disk at byte offset $20. Why not just give the calling process (or processes) a sector at a time, as they need it? In a word - fragmentation. That is, your file could end up being severely scattered over the disk's surface. This means that it takes more time to read it, as several seeks to different locations on the disk may have to be made. Also, you could potentially run out of segment map space. Briefly, each file and directory has a sector set aside for it that describes its characteristics. This is known as the 'file descriptor sector', or FD sector for short.

In the FD sector are data concerning the owner of the file, the dates of creation and last modification, the permissions, filesize, and lastly, a map containing a list of what sectors the file occupies on the disk. This map has 48 locations available. Each map location reserves three bytes for the logical sector number of the segment (LSN), and two bytes for the number of sectors in that segment. If the disk were reasonably full and highly fragmented, it is conceivable that you could fill up a segment list with 48 entries of one sector each - a 12K file would cause an error 217 (Segment List Full). In order to avoid this, the RBF manager tries to reserve a certain amount of sectors for file expansion each time a file needs more room.

The number of sectors to be added to the file is specified in IT.SAS. For most floppy disks, this number is 8. A good rule of thumb is to use some even multiple (or division) of the number of sectors available on a track. Standard OS9 floppies have 16 sectors per track, and I'm guessing that's why a default SAS for floppies is 8 sectors (one half of a track). I bet if they'd thought of it, they would have made the SAS value for Color Computer formatted disks 9, as that disk format has 18 sectors per track. Since most hard disks have 32 sectors per track (256 byte sectors), 16 and 32 are common values for IT.SAS in hard disk descriptors.

Allocating extra space to a file does not change the file's size. If the file is only occupying 35 bytes of the 8 sectors, its filesize will still show as 35 bytes. The rest of the 8 sectors is only standing by for use as expansion space for the file, but is not yet in use. It is however unavailable for use by any other file, and is marked as being busy in the disk's bitmap.

By now you're probably thinking that this is a horribly wasteful approach. Here's the catch: when the file is closed, the system will recapture all that expansion space and return it to the free space pool if the file pointer was at the end of the file. That is, if we had written the 35 byte file above, our write pointer would have moved forward one byte with each character we wrote to the file. When we wrote the 35th character, and then closed the file, we were at the end of file. As a result, the bulk of the reserved expansion space (in the case of a floppy, 7 sectors) was given back to the system.

This is wonderful for files that will be written or updated infrequently like ASCII text files written with your editor. If you think about it, you are not changing existing files, but writing out a new file and deleting the old one each time you edit. This is not so wonderful for files that are continually altered or expanded, like database files. These are the files that really need protection from fragmentation.

Fragmented database files (or directories) can make for extremely slow search or lookup operations. What can be done to effect this protection and hang onto the reserved expansion space? Simple - just move the file pointer to any place other than the end of file. This is normally done with a seek command, most commonly to location 0 in the file (also known as 'rewinding' a file). When the file is closed, whatever space had been allocated to it as expansion room is kept for future use, and not returned to the free space pool.

This technique is used by the system in the creation of directories. Each time 'MAKDIR' is run, it allocates the number of sectors specified in IT.SAS for that disk to the directory you create. Even though the filesize may be 64 (room for the '.' and '..' entries), the allocated space will be several sectors. This will prevent your directories from being scattered in several tiny chunks over the surface of your disks.

Could you imagine what the impact of having a 20 sector CMOS directory that was fragmented into 1 sector pieces? If you typed 'dire' instead of 'dir e', your heads would have to seek at least 20 times just to find out that there is no 'dire' command and issue an error 216.

There is a small disadvantage to this philosophy as well. If you are creating directories that you know in advance will only have a few files in them, that extra directory space will be permanently wasted. For this reason, I wrote a variation on the 'MAKDIR' command called 'mkdir' (Unix and MS-DOS users have one - why not us?).

Mkdir allows you to specify either the number of directory entries you wish to make room for, or just how many sectors you wish to reserve. Also available is the source for a utility called 'fmap' (file mapper). This utility was written primarily to allow inspection of the FD sectors, but is also useful in looking at a file's allocated space vice actual filesize, and its degree of fragmentation.

The sources to both of these programs are from early C programming efforts, and are consequently somewhat inelegant. If I let that stop me, I'd never publish or post anything, so here they are! For those without access to a C compiler, the binaries are available for downloading from the OS9 Forum on CompuServe in the UTILITIES area (DL9). [Source and binary files are available from the our PD Library .. ed]

I hope that I have managed to shed a little light on some of the less obvious areas of disk management, and possibly even demystified the disk bitmap and the nature of IT.SAS to some degree. I am always available in the OS9 Forum on CompuServe for questions and comments (as either 'Sysop' or 'Pete Lyall' - User Id: 76703,4230), and I check into the Delphi OS9 Forum at least weekly under the username 'OS9UGVP'.

For those with access to USENET, I can be reached at {scgvax,ihnp4,jplgodo}!wibr!pete or on the Internet at pete@wibr.eaton.com.

Please feel free to contact me with any questions you may have. Until the next time, happy hacking, and remember to backup your disks before getting too creative!

Document files and MAN.
or Access those docs easily!
by Bob Devries.

I have always felt that those of us who patronize the Public Domain Library, must have quite a hard job keeping track of where all those document files are. Those with hard disk drives, of course, can put them all together in one directory. But then to access them quite often means to have to type a rather long command line, especially if the directory is some way down the tree. Something like 'list /DD/USR/DOCS/ar.man'. And then you have to remember what is in the docs directory. A bit of a pain really.

Recently, I came accross a programme which solved the problem for me. It was an Amiga port of the UNIX programme 'MAN'. The programme, complete with document files and source code came from the 'Fred Fish' collection (a collection of some 460 disks of PD for the Amiga at 880K per disk). I thought it would be a good exercise to re-write the source code, which was, of course in C, for OS9. Well, it turned out to be relatively easy, as the code is not overly long.

The programme is quite versatile. You may specify the directory path for the documents, and also the programme with which to list them, on the command line. The programme's defaults are: 'list' for displaying the files and '/DD/MAN' for the directory to look for the document files. No special equipment is required to compile the programme (except for the C compiler, of course).

Command line options are as follows:

-v specify a new file viewer.
-f display the first manual page only.
-p add a path to the path list.
-d display path lists.

The programme can get extra path information from the file called 'manuals' which is kept in the same directory as the document files. You may also specify a viewer programme there to override the programme's defaults.

The 'manuals' file is laid out this way:

```
PATH /DD/MAN/
PATH /DD/SYS/
VIEW:more
START_MAN
ar                  /DD/MAN/ar.man
archive             /DD/MAN/archive.man
ed                  !ed /DD/MAN/eddocs.txt
END_MAN
```

There must be 20 characters and/or spaces between the first character of the programme name and the path to the document file inclusive. So in the first example:

```
ar                  /DD/MAN/ar.man
'--20 characters--'
```

You will notice that the third programme has an exclamation mark in its path list. This is done so that a completely different programme (ed in this case) will be used for that document only. Make sure, however, that the programme mentioned in that line is in the current execution directory, or in memory, or it will not run, and will not return any error (I might fix that later).

Now to the source code. You will notice that I have left the code in such a way that it should re-compile on the amiga still, as well as on the OS9 system. To compile, you should use the command line switch '-dos9' which has the same effect as a line in the source which says '#define os9'. So the compiler command line looks like this:

```
cc man.c -dos9
```

Without further ado, here's the source listing, which is of course also available from our PD library.

Regards,
Bob Devries

```
/*************************************************************************\
*                              Man V1.2o                                  *
*                Copyright (C) 1989 by G. Glendown                        *
*                                                                         *
* This program and all documentation is placed in the public domain.      *
```

```
 *No fee may be charged except for media and copying.  ArcPrep may be *
 * included on Fred Fish's distribution disk.  Any other PD-collector *
 * will have to get my permission prior to putting Man on his disks.  *
 *     Any net service may distribute Man, as long as no fee for      *
 *downloading is charged (exept normal line costs).  You may make any*
 * changes to Man, but I would appreciate it if you could send me the *
 *  new sources (except conversions to Lattice-C or Modula-II) or at  *
 *             least a short note about what you did...               *
 *                                                                    *
 * German PD dealers:                                                 *
 *            Please read the file 'GermanDistribution'               *
\********************************************************************/


/* ts=4 */
/*
        *       Man     V1.2
        *
        *       Unix-style manual command controlled completely by
        *       the file "man:manuals"
        *
        * USAGE:
        *
        *       man <manual> [-vTextViewer] [-pPath] [-f]
        *
*/

#include <stdio.h>

#define LINE    512
#define MAXCOM  16                      /* Maximum size of command name */
#define EMPTY   "                "
#define FNPOS   20                      /* start position of command line/filename */
#ifdef amiga
#define MANFILE "man:manuals"
#define EXIT exit(0L);
#endif
#ifdef os9
#define MANFILE "/DD/MAN/manuals"
#define EXIT exit(0);
#endif

/* Uncomment this if you don't have my library with a new PrintF... */
#define PrintF printf
int cnt,fc;

FILE *fh,*fopen();
char *fgets();

char line[LINE],w[MAXCOM<<1];
char path[1][128];
#ifdef amiga
char cmmnd[128]={"run less "};
#endif
#ifdef os9
char cmmnd[128]="list ";
#endif
```

```
int Status=0;
#define CUSTOMVIEW      (1<<0)
#define FIRSTONLY       (1<<1)
#define DISPLAYPATHS    (1<<2)
int pt;

char *name;

#ifdef os9
#define ACCESS_READ 1
#endif

/*****************************************************************************\
 *                                main                                       *
 *                                                                           *
 * Does all scanning of the CLI parameters, prints the paths defined  *
 *                   and checks the 'manuals'-file.                          *
\*****************************************************************************/

main(argc,argv)
int argc;
char *argv[];
{
char    *c;
int t;
#ifdef os9
        setbuf(stdout,NULL);
#endif
        if (argc<=1)
                PrintF("Use '%s man' to see usage of %s\n",argv[0],argv[0]),EXIT

    cnt=0;

        name=(char *)NULL;

        for (t=1;t<argc;t++) {
                if (argv[t][0]=='-') {
                        switch (toupper(argv[t][1])) {
                                case 'V':
                                        strcpy(&(cmmnd[4]),&(argv[t][2]));
                                        strcat(cmmnd," \0");
                                        Status =CUSTOMVIEW;
                                        break;
                                case 'P':
                                        strcpy(path[pt],&argv[t][2]);
                                        pt++;
                                        break;
                                case 'F':
                                        Status =FIRSTONLY;
                                        break;
                                case 'D':
                                        Status =DISPLAYPATHS;
                                        break;
                                default:
                                        PrintF("Illegal option '%c' ignored\n",argv[t][1]);
                                        break;
                        }
```

```
                }
                else
                        name=argv[i];
        }

        if ((name==(char *)NULL)&&(!(Status&DISPLAYPATHS)))
                PrintF("No manual specified!\n"),EXIT

        if (fh=fopen(MANFILE,"r"))
        {
                SearchStart(fh);
                if (Status&DISPLAYPATHS) {
                        PrintF("Man-Paths:\n");
                        for (t=0;t<pt;t++)
                                PrintF("    %s\n",path[t]);
                        PrintF("File-viewer: '%s'\n",cmmnd);
                        fclose(fh);
                        EXIT
                }
                strncpy(w,name,MAXCOM);
                strcat(w,EMPTY);
                do {
                        if (!(c=fgets(line,LINE,fh)))
                                PrintF("Oops! Manual-file corrupt!\n"),ex(name);
                        if (strncmp(line,w,strlen(w))==0) {
                                cnt++;
                                fc++;
                                DisplayFile(&line[FNPOS]);
                                if (Status&FIRSTONLY) {
                                        fclose(fh);
                                        EXIT
                                }
                        }
                        if (strncmp(line,"END_MAN",7)==0) ex(name);
                }
                while (1);
        }
        ex();
}

/*****************************************************************************\
 *                                ex                                         *
 *  This routine looks through all defined paths to find a file that  *
 *                  will match the filename given.                           *
 *****************************************************************************
 * Parameters:                                                               *
 * - name of the manual file  (char *)                                       *
\*****************************************************************************/

ex(t)
char *t;
{
int g;
        if (cnt==0) {
#ifdef amiga
                if (Found("man:",t)) {
                        strcpy(line,"man:");
```

```
#endif
#ifdef os9
                if(Found("/DD/MAN/",t)) {
                        strcpy(line,"/DD/MAN/");
#endif
                        strcat(line,t);
                        DisplayFile(line);
                        if (Status&FIRSTONLY) goto TheEnd;
                }
                for (g=0;g<pt;g++) {
                        if (Found(path[g],t)) {
                                strcpy(line,path[g]);
                                strcat(line,t);
                                DisplayFile(line);
                                if (Status&FIRSTONLY) g=pt;
                        }
                }
        }

        if (!fc) PrintF("No manuals available!\n");

TheEnd:
        if (fh) fclose(fh);
        exit(0);
}

/*****************************************************************************\
 *                               Found                                       *
 *  This routine checks, if the file is accessable in the directory          *
 *                       given as a parameter                                *
 *****************************************************************************
 * Parameters:                                                               *
 * - path which is to be searched  (char *)                                  *
 * - filename to look for (char *)                                           *
 *****************************************************************************
 * Result:                                                                   *
 *  >!< if file is found                                                     *
 *  >0< file not found                                                      *
\*****************************************************************************/

int Found(p,n)
char *p,*n;
{
char f[256];
#ifdef amiga
struct FileLock *l,*Lock();
#endif
#ifdef os9
        int l;
#endif
        strcpy(f,p);
        strcat(f,n);
#ifdef amiga
        l=Lock(f,ACCESS_READ);
        if (l) UnLock(l),fc++;
        if (l) return(l);
        else return(0);
```

```
#endif
#ifdef os9
        i=access(f,ACCESS_READ);
        if (i == -1) return(0);
        else return(1);
#endif
}

/*************************************************************************\
 *                          DisplayFile                                  *
 * This routine will call either the file viewer, or it will execute     *
 *                     the command supplied.                             *
 *************************************************************************
 * Parameters:                                                           *
 * - line  includes either the filename or the line to be executed       *
\*************************************************************************/

DisplayFile(f)
char *f;
{
int t;
char exec[256];
        t=0;
        if (*f=='!')
                Execute(&f[1],0L,0L);
        else {
                do {
                        t++;
                } while ((f[t]!=' ')&&(f[t]!='\n'));
                strcpy(exec,cmmnd);
                strncat(exec,f,t);
                PrintF("%s - ",f);
                Execute(exec,0L,0L);
        }
}

/*************************************************************************\
 *                          SearchStart                                  *
 *This routine scans the 'manuals'-file for the PATH and VIEW commands*
 *            and tries to find the 'START_MAN'-command.                 *
 *************************************************************************
 * Parameters:                                                           *
 * - filepointer to the 'manuals'-file.                                  *
\*************************************************************************/

SearchStart(fh)
struct FileHandle *fh;
{
char line[LINE],*c;
        for (;;) {
                if (c=fgets(line,LINE,fh)) {
                        if (strncmp(line,"START_MAN",9)==0) return;
                        if (strncmp(line,"VIEW:",5)==0) {
                                if ((Status&CUSTOMVIEW)==0) {
                                        strcpy(cmmnd,&line[5]);
                                        cmmnd[strlen(cmmnd)-1]=' ';
                                }
```

```
                        }
                        if (strncmp(line,"PATH ",5)==0) {
                                if (pt<10) {
                                        strcpy(path[pt],&line[5]);
                                        path[pt][strlen(path[pt])-1]='\0';
                                        pt++;
                                }
                                else
                                        PrintF("Too many paths in 'man:manuals' - file!\n");
                        }
                }
        }
}

#ifdef os9
Execute(exec,dummy1,dummy2)  /* this required to replace function */
                             /* non-existant in os9 */
char *exec;
long dummy1;
long dummy2;
{
        system(exec);
}
#endif
```

oooooooooo0000000000oooooooooo